SME0892 - Cálculo Numérico para Estatística

Trabalho 1 - Velocidade de resolução de sistemas lineares tridiagonais

Carolina Spera Braga - Número USP: 7161740

20 de maio, 2022

Introdução

O objetivo desta atividade foi resolver sistemas lineares da forma $A \cdot x = \text{b}$ e comparar a velocidade de processamento de vários métodos para uma classe especial de sistemas lineares, cuja matriz é simétrica positiva definida (SPD) e tridiagonal.

As matrizes utilizadas nesta atividade satisfazem ainda o critério:

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|$$

e são chamadas de matrizes diagonais estritamente dominantes, ou seja, o coeficiente da diagonal em cada equação deve ser maior que a soma dos valores absolutos dos outros coeficientes da equação.

Os métodos avaliados foram os seguintes:

- Usando diretamente o comando *linalg.solve()* do *Python*;
- Decomposição LU;
- Decomposição de Cholesky;
- Eliminação de Gauss sem pivoteamento;
- Método de Gauss-Jacobi;
- Método de Gauss-Seidel.

Sendo os quatro primeiros métodos diretos e os dois últimos métodos iterativos. Os métodos de Gauss-Jacobi e Gauss-Seidel foram implementados com uma tolerância inferior a 10^{-8} como critério de parada, e avaliados também para uma tolerância de 10^{-2}.

As matrizes tridiagonais SPD foram padronizadas e avaliadas nas dimensões 64, 128, 256, 512, 1024, 2048, 4096, 8192 e 16384, sendo 16384 o máximo valor possível rodado pelo meu notebook pessoal.

A solução do sistema $A \cdot x = \text{b}$ para as decomposições LU e de Cholesky foram calculadas diretamente pelas funções *lu solve* e *cho solve*, respectivamente, mas os códigos para decompor primeiro a matriz A em $L \cdot U$ e A em $H \cdot H^T$ e em seguida aplicar substituições progressivas e regressivas para encontrar o valor de x também se encontram no arquivo *7161740-codigo.py*. No mesmo arquivo também se encontra uma solução para ambas as decomposições através do comando *dot()*. O código utilizado de fato foi apenas a solução mais simples.

Gráficos

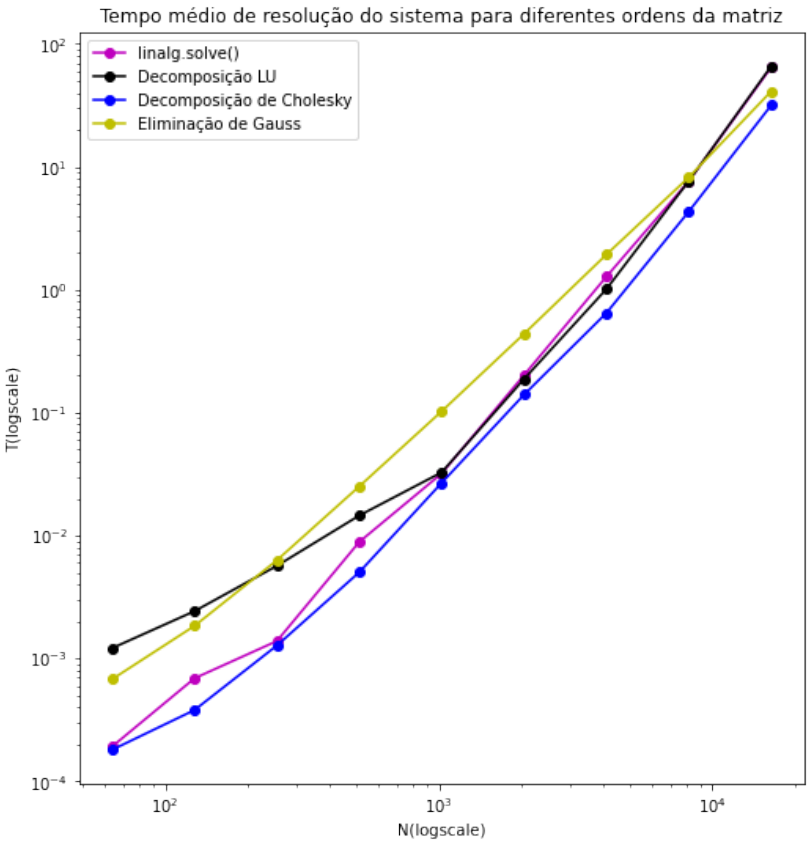

Figura 1: Tempo médio de resolução do sistema $A \cdot x = \mathrm{b}$ para diferentes ordens da matriz para os métodos diretos em escala logarítmica.

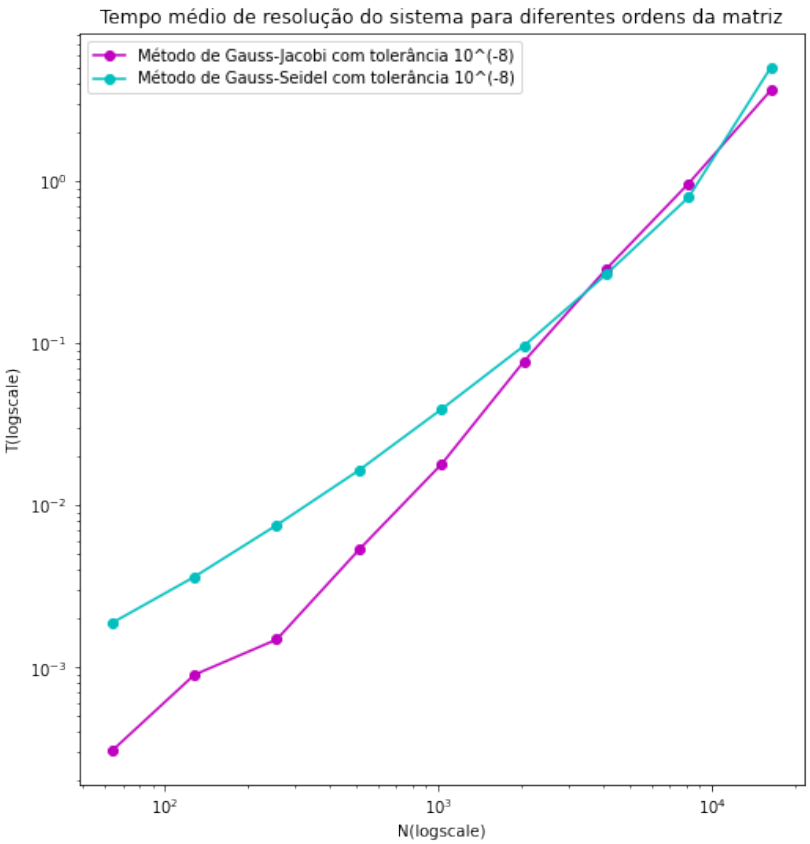

Figura 2: Tempo médio de resolução do sistema $A \cdot x = \mathrm{b}$ para diferentes ordens da matriz para os métodos iterativos em escala logarítmica, com tolerância de 10^{-8}.

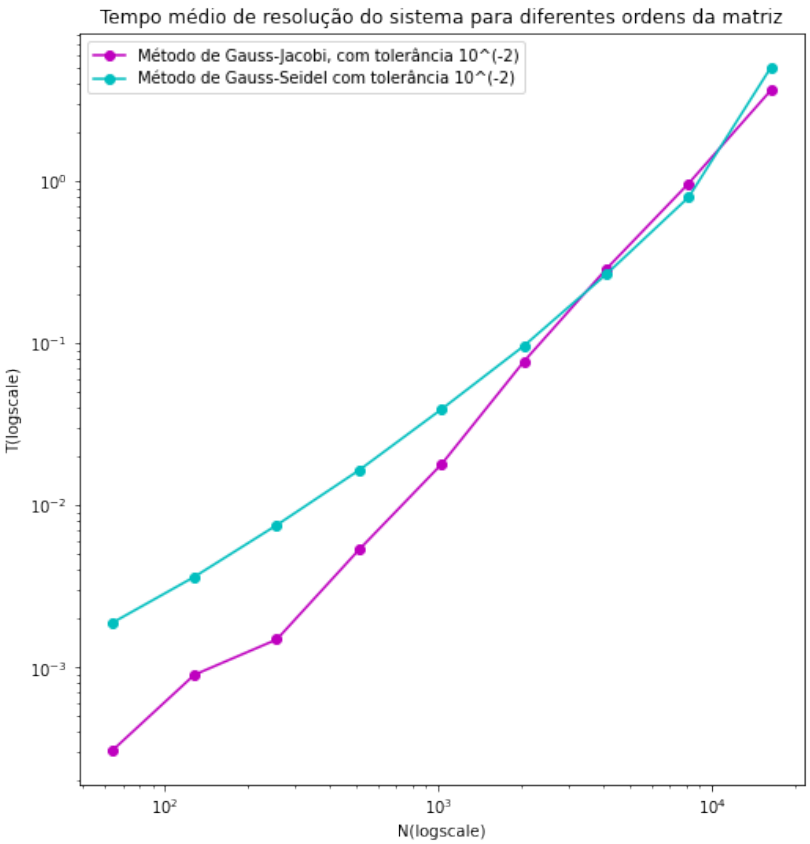

Figura 3: Tempo médio de resolução do sistema $A \cdot x = \mathrm{b}$ para diferentes ordens da matriz para os métodos iterativos em escala logarítmica, com tolerância de 10^{-2}.

Resultados

Os tempos de processamento de cada método para cada dimensão da matriz foram obtidos através de uma média de pelo menos 10 tempos.

Para verificar como o tempo de resolução do sistema $A \cdot x = \mathrm{b}$ cresce conforme a dimensão da matriz aumenta podemos comparar os gráficos das Figuras 1 e 2 com a ordem de custo computacional.

O cálculo da ordem de custo computacional pode ser obtido através equação $\mathrm{T(N)} = C \cdot N^p$, onde T é um vetor com os valores do tempo de processamento, N é um vetor com os valores das ordens das matrizes, C é uma constante real e p é o valor da ordem de custo, e também podemos obtê-lo pela função *np.polyfit()* do *Python*. Os valores aproximados obtidos por *np.polyfit()* estão listados a seguir:

- 2.7531 para o Método 1 - *linalg.solve()*
- 2.8282 para o Método 2 - Decomposição LU
- 2.6205 para o Método 3 - Decomposição de Cholesky
- 2.1749 para o Método 4 - Eliminação de Gauss
- 1.8430 para o Método 5 - Método de Gauss-Jacobi com limite de tolerância 10^{-8}
- 1.8705 para o Método 6 - Método de Gauss-Seidel com limite de tolerância 10^{-8}
- 1.7745 para o Método 5 - Método de Gauss-Jacobi com limite de tolerância 10^{-2}
- 1.8503 para o Método 6 - Método de Gauss-Seidel com limite de tolerância 10^{-2}

A partir da análise gráfica e dos valores calculados da ordem de custo computacional, percebemos um crescimento aproximadamente cúbico para os métodos 1, 2 e 3, e um crescimento aproximadamente quadrático para os métodos 4, 5 e 6 do tempo de resolução do sistema $A \cdot x = \mathrm{b}$ conforme aumentamos a dimensão da matriz A.

Foram testados 4 códigos diferentes para o algorítmo de Gauss-Seidel e, teoricamente, a velocidade de processamento do método de Gauss-Seidel deveria ser maior do que a do método de Gauss-Jacobi, mas os algorítimos implementados retornaram o oposto. Portanto, nessa atividade, com estes algorítmos contidos no arquivo *7161740-codigo.py* e para as dimensões da matriz A entre 64 e 16384, temos que o método de Gauss-Jacobi é mais rápido que o método de Gauss-Seidel.

Ainda com relação aos métodos iterativos, teoricamente o método de Gauss-Seidel converge para a solução do sistema mais rápido que o método de Gauss-Jacobi, mas caso o sistema não tenha uma convergência para a solução tão rápida tendo em vista o limite de tolerância definido de 10^{-8}, e considerando que limitamos o código a um limite máximo de iterações iguais para os dois métodos, talvez seja essa a explicação para o fato de o código do método de Gauss-Seidel levar um tempo maior de processamento que o método de Gauss-Jacobi, ou seja, como não convergem para a solução, ambos calculam todas as iterações permitidas e disto resulta a incongruência relatada.

Tendo em vista o que foi dito no parágrafo anterior, realizou-se todo o processo para os métodos iterativos mas com limite de tolerância igual a 10^{-2}, neste caso temos uma convergência clara dos métodos, e portanto temos resultados em acordo com a teoria, ou seja, conforme aumentamos a dimensão da matriz A, percebemos que o Método de Gauss-Seidel é mais rápido que o método de Gauss-Jacobi. Os resultados aproximados deste procedimento para o custo computacional usando a função *np.polyfit()* do *Python* são:

- 1.7745 para o Método 5 - Método de Gauss-Jacobi
- 1.8503 para o Método 6 - Método de Gauss-Seidel

No gráfico representado na Figura 3 temos uma visualização do crescimento mais rápido do tempo de processamento do método de Gauss-Jacobi em relação ao Gauss-Seidel com o limite de tolerância 10^{-2}. E embora os gráficos das Figuras 2 e 3 sejam muito similares, quando calculamos a projeção no tempo para cada método encontramos diferenças na velocidade de processamento entre eles, o que foi observado na questão 2 da seção seguinte. Os valores dos tempos obtidos podem ser verficados no arquivo *7161740-codigo.py*.

Perguntas e respostas

1. É possível estimar uma ordem de custo para os métodos iterativos?

   Sim, como mostrado acima, podemos usar tanto a equação $\mathrm{T(N)} = C \cdot N^p$ quanto a função *np.polyfit()* do *Python*. Os resultados aproximados obtidos em *Python* para os métodos iterativos com ambos os limites de tolerância calculados estão listados a seguir:

   - 1.8430 para o Método 5 - Método de Gauss-Jacobi com limite de tolerância 10^{-8}
   - 1.8705 para o Método 6 - Método de Gauss-Seidel com limite de tolerância 10^{-8}
   - 1.7745 para o Método 5 - Método de Gauss-Jacobi com limite de tolerância 10^{-2}
   - 1.8503 para o Método 6 - Método de Gauss-Seidel com limite de tolerância 10^{-2}

2. Seguindo a tendência de crescimento observada de cada método, estime quanto tempo cada método demoraria para resolver um sistema desses com 1 milhão de incógnitas no seu computador? E com 1 bilhão de incógnitas?

   O código utilizado para responder esta pergunta se encontra no arquivo *7161740-codigo.py*.

   Para N = 1 milhão, temos:

   - *linalg.solve()*: 338996.3697 s $\approx$ 3.92 dias
   - Decomposição LU: 349844.0846 s $\approx$ 4.05 dias
   - Decomposição de Cholesky: 161079.2439 s $\approx$ 1.86 dias
   - Eliminação de Gauss: 176986.5722 s $\approx$ 2.05 dias
   - Método de Gauss-Jacobi com limite de tolerância 10^{-8}: 212.1316 s $\approx$ 3.54 minutos
   - Método de Gauss-Seidel com limite de tolerância 10^{-8}: 282.6500 s $\approx$ 4.71 minutos
   - Método de Gauss-Jacobi com limite de tolerância 10^{-2}: 111.9015 s $\approx$ 1.87 minutos

- Método de Gauss-Seidel com limite de tolerância 10^{-2}: 108.3808 s $\approx$ 1.81 minutos

Para N = 1 bilhão, temos:

- *linalg.solve()*: 340722624026.168 s $\approx$ 10796.85 anos
- Decomposição LU: 351665863039.3879 s $\approx$ 11143.62 anos
- Decomposição de Cholesky: 161829479651.0073 s $\approx$ 5128.07 anos
- Eliminação de Gauss: 177403317702.246 s $\approx$ 5621.57 anos
- Método de Gauss-Jacobi com limite de tolerância 10^{-8}: 212347.6092 s $\approx$ 2.46 dias
- Método de Gauss-Seidel com limite de tolerância 10^{-8}: 282981.6718 s $\approx$ 3.28 dias
- Método de Gauss-Jacobi com limite de tolerância 10^{-2}: 112009.6959 s $\approx$ 1.3 dias
- Método de Gauss-Seidel com limite de tolerância 10^{-2}: 108506.7025 s $\approx$ 1.26 dias

Novamente, podemos ver que para os métodos iterativos com limite de tolerância 10^{-8}, o método de Gauss-Jacobi é mais rápido e quando o limite de tolerância é 10^{-2}, o método de Gauss-Seidel é o mais eficiente.

3. De acordo com os resultados, qual tipo de método seria mais apropriado para este tipo de problema? Por quê?

   O método mais apropriado para esse problema, da forma como foi proposto com limite de tolerância para os métodos iterativos de 10^{-8} e para as dimensões da matriz A entre 64 e 16384, seria o Método de Gauss-Jacobi, pois foi o que apresentou os menores tempos de forma geral, inclusive com o crescimento da dimensionalidade da matriz se manteve um algorítmo rápido, e foi o que apresentou menor ordem de custo computacional.

4. Dentre de cada grupo, qual método iterativo e qual método direto apresentou melhor performance?

   Dentre os métodos iterativos, para um limite de tolerância de 10^{-8}, o Método de Gauss-Jacobi apresentou a melhor performance com menores tempos de processamento e menor ordem de custo computacional, e quando analisamos um limite de tolerância de 10^{-2} o método de Gauss-Seidel se mostra mais rápido, porém com ordem de custo computacional maior que a do método de Gauss-Jacobi. E dentre os métodos diretos a Decomposição de Cholesky foi o algorítmo mais rápido, e a Eliminação de Gauss apresentou a menor ordem de custo computacional.